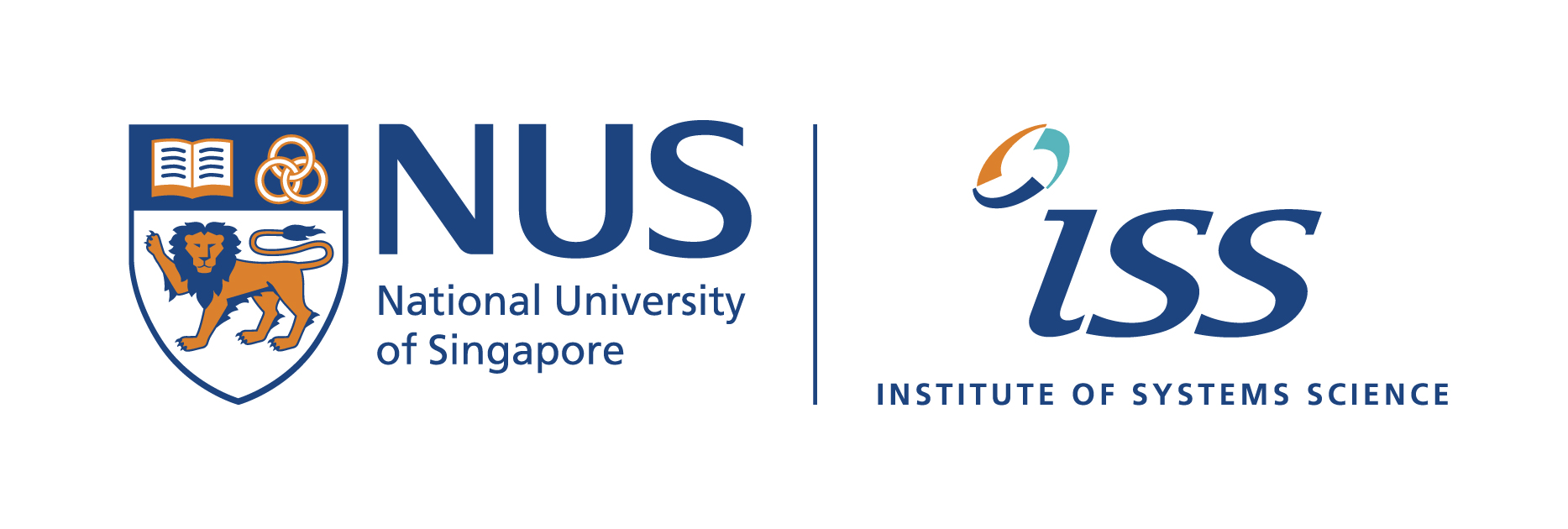
****

**EB5206 Computational Intelligence**:

**Regression Assignment**

**To Predict the housing price of NYC by Neural Network Models**

**Lecturer:** Dr. ZHU Fang Ming

Dr. TIAN Jing

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Name** | **E-mail** |
| A0178222H | Zhu Huiying | [e0267533@u.nus.edu](mailto:e0267533@u.nus.edu) |
| A0178431A | Huang Qingyi | [e0267742@u.nus.edu](mailto:e0267742@u.nus.edu) |
| A0178415Y | Jiang Zhiyuan | [e0267726@u.nus.edu](mailto:e0267726@u.nus.edu) |

# Table of Contents

[Table of Contents 0](#_Toc514104007)

[1. Introduction 1](#_Toc514104008)

[2. Data Preparation 1](#_Toc514104009)

[2.1 Selection of Attributes 1](#_Toc514104010)

[2.2 Log-Transform of Target Variable 3](#_Toc514104011)

[2.3 Data coding 4](#_Toc514104012)

[2.4 Implementation on Missing Values 4](#_Toc514104013)

[2.5 Separation of data 4](#_Toc514104014)

[2.6 Standardization of data 4](#_Toc514104015)

[2.7 Removal of Outliers 5](#_Toc514104016)

[3. MLFF-BP Neural Network Model 5](#_Toc514104017)

[3.1 Model Introduction 5](#_Toc514104018)

[3.2 Model Building 6](#_Toc514104019)

[3.3 MLFF-BP Model Performance 8](#_Toc514104020)

[3.4 MLFF-BP Model Improvement 9](#_Toc514104021)

[4. Radial Basis Function (RBF) Neural Network Model 11](#_Toc514104022)

[4.1 Model Introduction 11](#_Toc514104023)

[4.2 Model Building 11](#_Toc514104024)

[4.3 RBF Model Performance 12](#_Toc514104025)

[5. Models Ensemble 13](#_Toc514104026)

[6. Conclusion 14](#_Toc514104027)

Appendix A

# Introduction

The Brooklyn Housing dataset, primarily from NYC Housing Sales Data Found, provided home buyers with much more influences price negotiations than the number of bedrooms or a white-picket fence. This dataset for 2017 contains 23955 housing price samples and comprises of 108 explanatory variables describing almost every aspect of residential homes in Brooklyn.

# Data Preparation

Prior to using the dataset for analysis, a series of data preparation activities were carried out. This includes a sanity check of the data reported for each variable to ensure that the range of values reported for each variable was logical. The steps involved in data preparation are detailed in the following sub sections.

## Selection of Attributes

As identified in the data dictionary (included in the Appendix), “sale\_price” was chosen as the target. Among the remaining 109 explanatory variables, 11 geospatial variables such as “XCoord”, “YCoord” and “Address” and 8 variables with unique values like “OwnerName” and “sale\_date” were removed, since this case mainly focused on internal features influenced the housing price. In addition, 19 variables which have more than 90% same values were moved for small variance they carry, and 24 variables containing a large amount of missing value were discarded as well. After that, a dataset contains 47 variables with 44 numeric variables and 3 categorical variables was selected.

In order to enhance the predictive performance, the correlations between each numeric variable and sale price were inspected by calculation correlation coefficient. To narrow our focus on explanatory variables that have a high impact on the target variable, variables with an absolute correlation coefficient value higher than 0.25 were selected for model building. The correlation level between these 20 selected numeric variables and sales price can be observed in Figure 1 shown below.

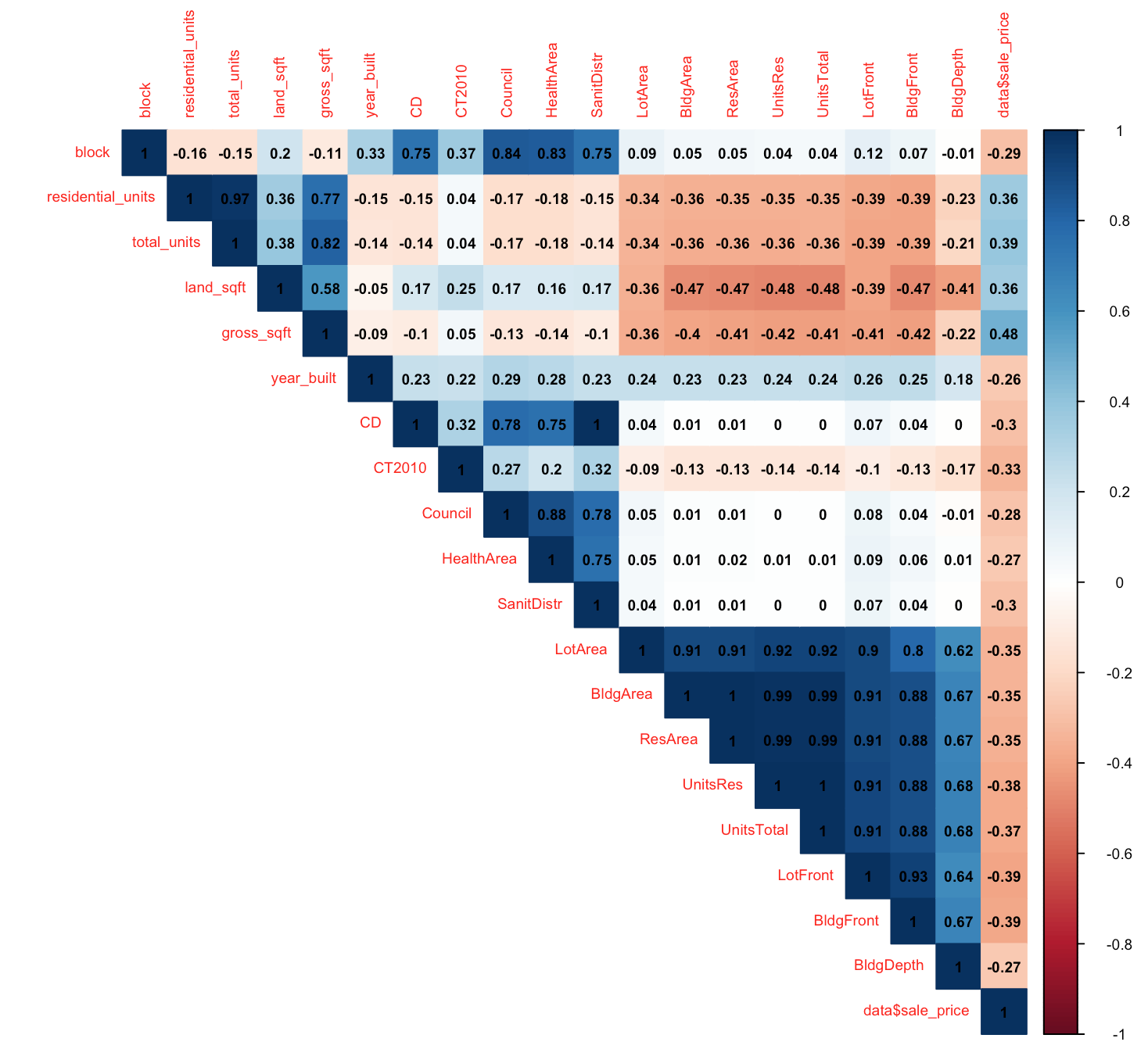


Figure 1: Correlation Matrix versus Sale Price

As such, 25 numeric variables and 3 categorical variables were remained for the further prediction process, which show in Table 1 below.

Table 1: Selected Variables

|  |  |  |
| --- | --- | --- |
| **Remained Variable Name** | **Type** | **Count** |
| sale\_price | taget | 1 |
| tax\_class; SplitZone; IrrLotCode | categorical | 3 |
| block; residential\_units; total\_units; land\_sqft gross\_sqft; year\_built; CD; CT2010; Council; HealthArea; SanitDistr; LotArea;  BldgArea; ResArea; UnitsRes; UnitsTotal;  LotFront; BldgFront; BldgDepth; AssessLand;  AssessTot; YearBuilt; BBL; SHAPE\_Leng; SHAPE\_Area; | numeric | 25 |

## Log-Transform of Target Variable

For a more precise prediction, the value of target variable “sale\_price” was narrowed to a range from 100,000 to 3000,000 with 16366 observations remained, based on domain knowledge and an exploration using boxplot. Then a histogram of the distribution of sales price was plotted for a better visualization. From Figure 2 below, a highly right-skewed distribution was observed, which indicated that there were some extremely high values of sale price away from the center value. To reduce the impact of these observations, a log-transform was implemented. After this transform, a nearly normal distribution of sale price can be gained, shown in Figure 3.

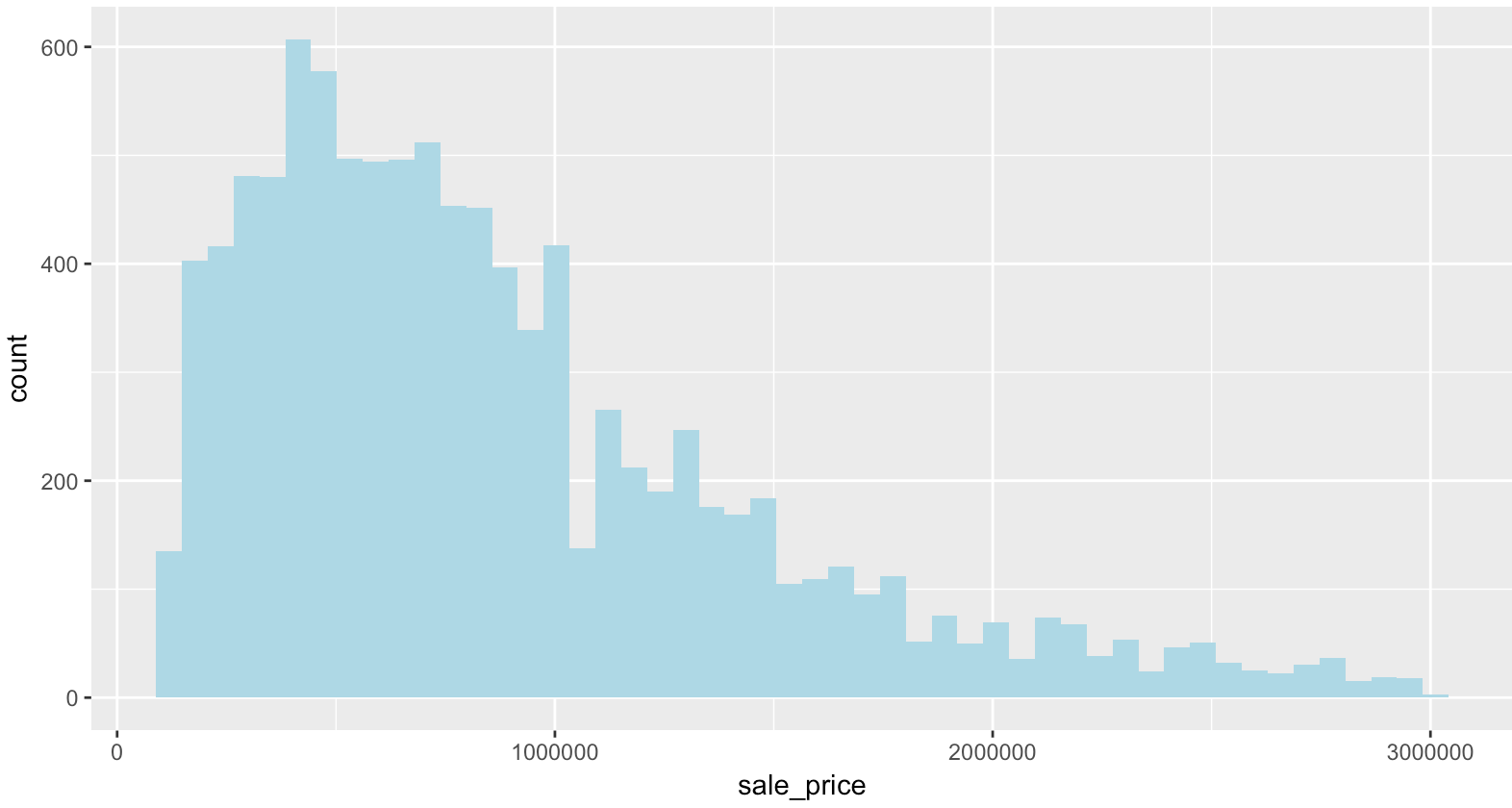


Figure 2: Histogram of Sale Price, before Log-Transform

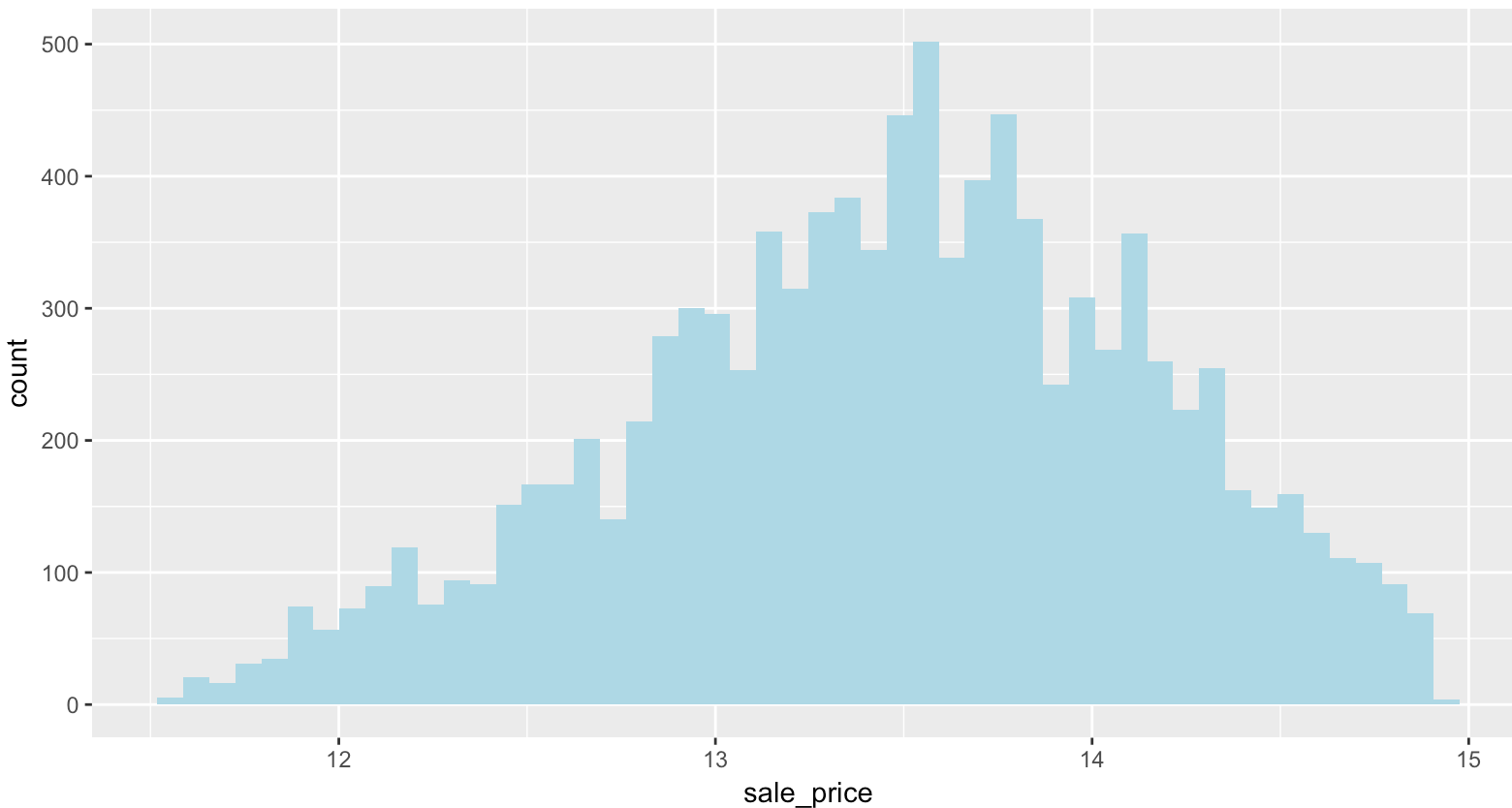


Figure 3: Histogram of Sale Price, after Log-Transform

## Data coding

It is noted that data is required to be prepared in specific ways before fitting in a machine learning model, as all input variables are needed to be numeric. For categorical variables where no ordinal relationship exists, a one-hot encoding can be applied, by which a new binary variable is added for each unique category. As such, 3 categorical variables were encoding into 12 new binary variables, while each category of ordinal variables was represented by an integer value.

## Implementation on Missing Values

After variables selection and data encoding, missing values were inspected using R and 6248 observations containing “NA” were detected. For these “NA” contained observations, more than half of the dependent variables have missing values, which means that such observation was lack of indicators for sales price prediction. As such, all these 6248 observations were removed, while 10118 observations remained after this step.

## Separation of data

To evaluate the performance of neural network models, the whole dataset was randomly split into training and test datasets with a ratio of 7:3. Further, 20 percent of the training data was split out for validation, a process to detect overfitting, while the remaining 80 percent was used for model training. The test dataset was used to test the generalization ability of the model using new data.

Table 2: Size of Separation Dataset

|  |  |
| --- | --- |
| **Usage** | **number of observations** |
| **total** | **10118** |
| training | 5666 |
| validation | 1416 |
| test | 3036 |

## Standardization of data

Prior to fitting neural networks predictive models, rescaling is an important step for parameters of different units and scales. There are two methods usually well known for rescaling, Normalization and Standardization. To avoid scaling “normal” data to a very small interval as outliers always exist in most of datasets, Standardization instead of Normalization was used in this case, by which data was transformed to have zero mean and unit variance, based on the equation below.

As one important assumption in modelling is that feature in training dataset would be the same in testing set, test dataset was scaled using the mean and standard deviation of training data.

## Removal of Outliers

To gain a better predictive ability of the model, outliers of numeric variables were need to be excluded. According to Figure 3 below, observations with an absolute standardized higher than 4 were removed. As such, 8712 observations were remained for training and testing purpose. In addition, to guarantee the evaluative ability of the test dataset, prior to the model building stage, processes of removing outliers only applied on training and validation dataset.

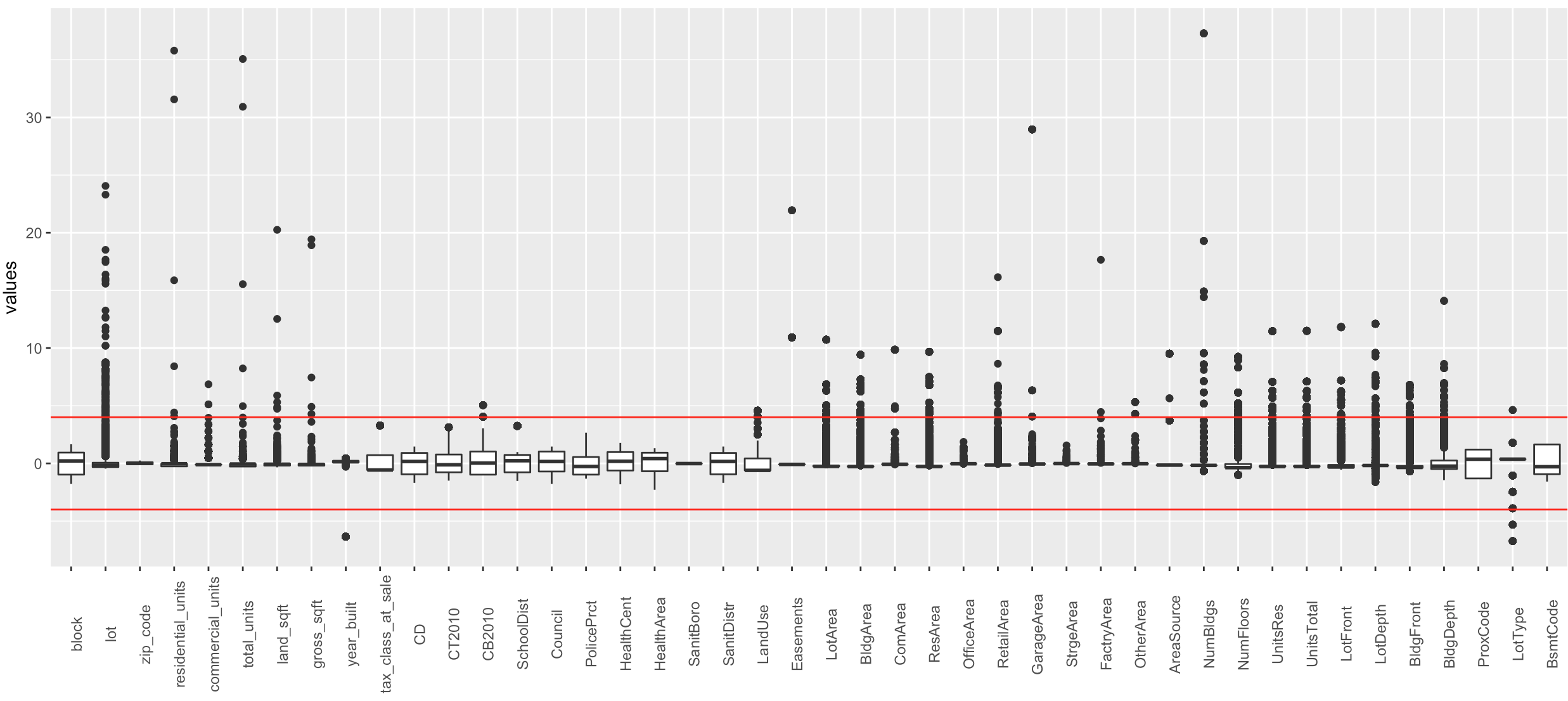


Figure 4: Boxplot of Numeric Variables, after Standardization

# MLFF-BP Neural Network Model

## Model Introduction

MLFF-BP, Multilayer Feedforward with Backpropagation is a type of neural network using dot products (between inputs and weights) and sigmoid activation functions (or other monotonic functions), while training through backpropagation learning rules.

## Model Building

To gain a better performance of the MLFF-BP model, properties showed in Table 2 are needed to be considered.

Table 3: MLFF-BP Properties

|  |  |
| --- | --- |
| **MLFF-BP Properties** | **Value** |
| activation function | ReLU |
| optimizer | Adam |
| loss function | MSE |
| dropout rate | 0.1 |
| number of epochs | 300 |
| batch size | 100 |
| number of hidden nodes | 6 |
| number of hidden layers | 1 |
|  |  |
|  |  |

Compared to the most common used activation function “Sigmoid”, “ReLU” was chosen in this case, for two major benefits of this activation function, sparsity and a reduced likelihood of vanishing gradient. To have a better understanding of two major benefits of “ReLU”, the definition of “ReLU” is showed below:

It is noted that sparsity arises when a≤0. The more such units that exist in a layer the much sparse the resulting representation. “Sigmoid” are always likely to generate some non-zero value resulting in dense representations. Sparse representations seem to be more beneficial than dense representations. On the other hand, when a>0, the likelihood of the gradient to vanish increases. As such, the gradient has a constant value. In contrast, the gradient of “Sigmoid” becomes increasingly small as the absolute value of x increases. Therefore, the constant gradient of “ReLU” results in faster learning.

As gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks, choosing the suitable algorithm for optimizing gradient descent is curial. Adaptive Moment Estimation (Adam) is a method that computes adaptive learning rates for each parameter, which keeps an exponentially decaying average of past gradients, similar to momentum. Compared to the most common used optimizer SGD, Adam can effectively reduce training time and has a higher convergence rate. Therefore, it was chosen as optimizer in this predictive model.

About the loss function, both MAE and MSE express average model prediction error, where lower values are better. However, MSE is more sensitive to error outliers, since the errors are squared before averaging. This means the MSE should be more useful when large errors are particularly undesirable. As such in this case, MSE was chosen as loss function.

It is noted that overfitting is one of the severe issues needed to be avoided when training neural network models. The dropout function is designed against overfitting by randomly omit a certain percentage of the hidden neurons during training. However, the drawback of the dropout function is a decrease in the accuracy of training dataset. In this case, a dropout rate around 0.1 was identified to serve the model well.

Since building a neural network model is an iterative process based on gradient descent, with a limited dataset, updating the weights with a singles pass is not enough. As the number of epochs increases, more number of times the weight are changed in the neural network, resulting in the model changing from underfitting to optimal to overfitting. After 10 experiments, increasing the number of epochs from 100 to 500, the optimal model was observed with 300 epochs.

As an important parameter, the number of hidden nodes is empirically chosen based on “Timothy Masters” rules below, where “nin” and “non” are respectively the number of input and output nodes in the model. To verify this empirical rule, MLFF-BP networks with four different combination of hidden layers and hidden nodes were built. As Table 4 shown below, the best model performance occurred when model contains one hidden layer with 6 hidden nodes. As such, this theory was verified and a model with 36 inputs, one hidden layer containing 6 hidden nodes and one output were built.

Table 4: MLFF-BP Performance

|  |  |  |  |
| --- | --- | --- | --- |
| **Network Stucture** | **training loss** | **validation loss** | **test loss** |
| 36-18-9-1 | 0.1960 | 0.1810 | 0.1947 |
| 36-6-3-1 | 0.1967 | 0.1784 | 0.1931 |
| 36-4-2-1 | 0.2023 | 0.1811 | 0.1988 |
| 36-6-1 | 0.1900 | 0.1820 | 0.1893 |

Based on the properties set above, a MLFF-BP network model was built. The structure of this “36-6-1” MLFF\_BP shows in Figure 5 below:

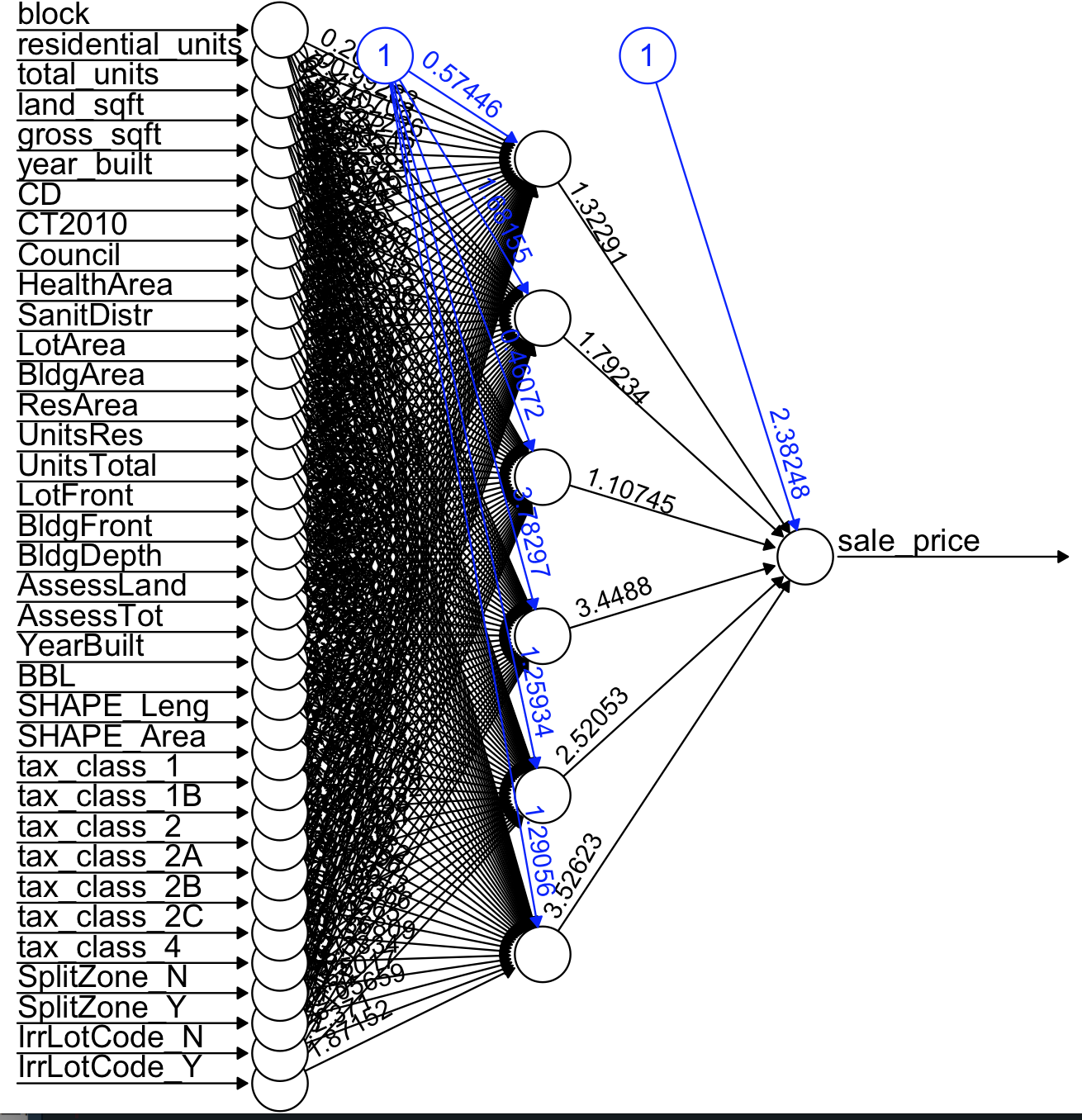


Figure 5: 36-18-9-1 MLFF-BP Structure

## MLFF-BP Model Performance

As Figure 6 shown below, the MSE of the training dataset after 300 epochs training converged around 0.19, while the error of validation dataset was relatively lower, due to the dropout step applied on each hidden layer. Compared the two error lines of training and validation dataset, there was no indication of overfitting. As such, test data was then fitted into the model to test its predictive ability. From Table 4 above, the loss of test dataset is 0.1893, with a 2% increase of the MSE value, which was considered as an acceptable result.

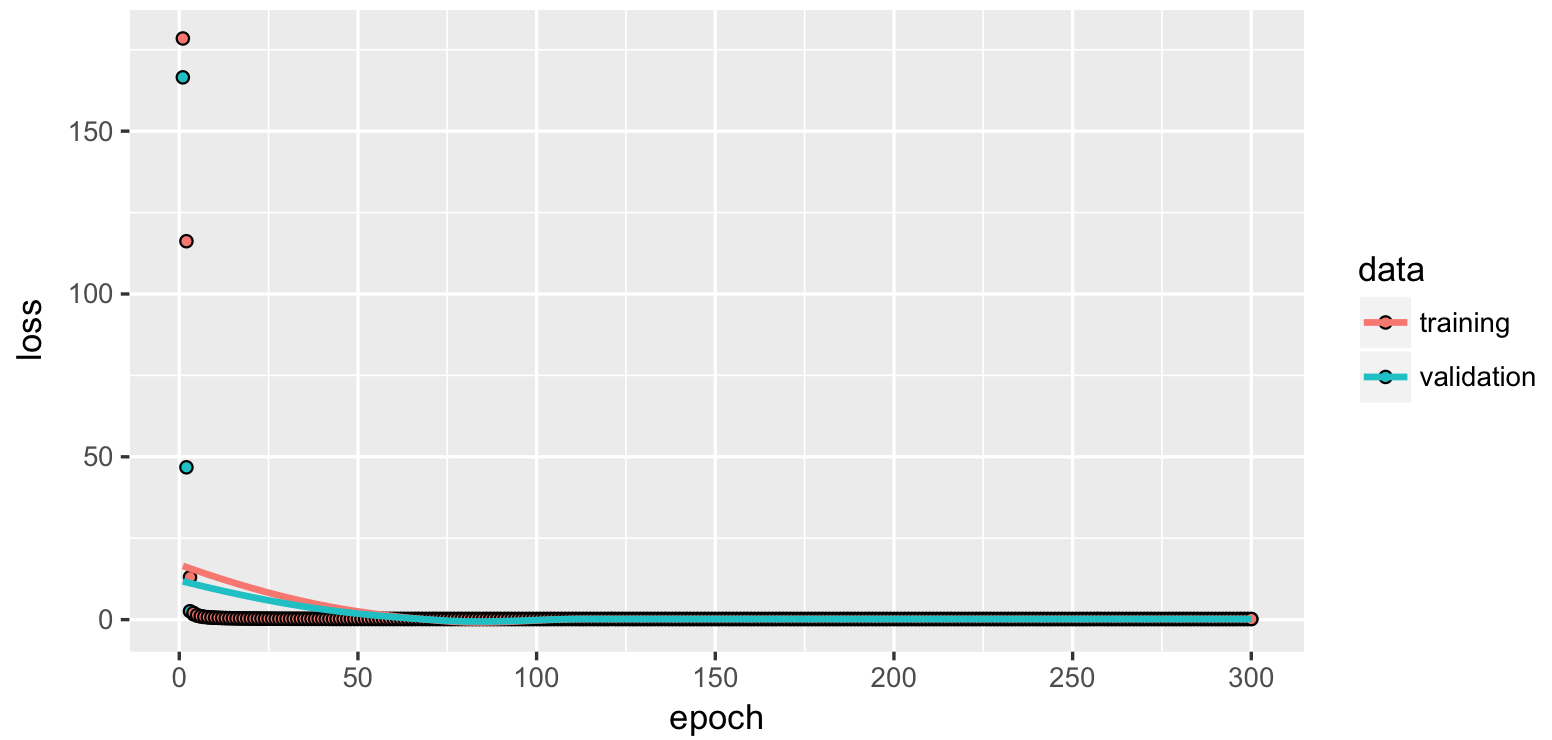


Figure 6: Loss by epochs

## MLFF-BP Model Improvement

It is noted that when the number of weights in neural network increases, the amount of data needed to be able to reliably determine the weights of the network also increase rapidly, which means that over-fitting is more likely to become more of an issue. As one solution to this problem, Principle Component Analysis, PCA, as a dimension reduction method, can be used to reduce the size of the network, and hence reduce the amount of data needed to train the network. However, one drawback of implementation PCA is that discriminative information might be in the low variance components, which might make performance worse.

From Figure 7 and Table 5 below, 9 principle components were extracted by Principle Component Analysis to achieve the balance between dimension reduction and the loss of variance. These 9 principle components cover 96% total variance, with only 4% variance lost.

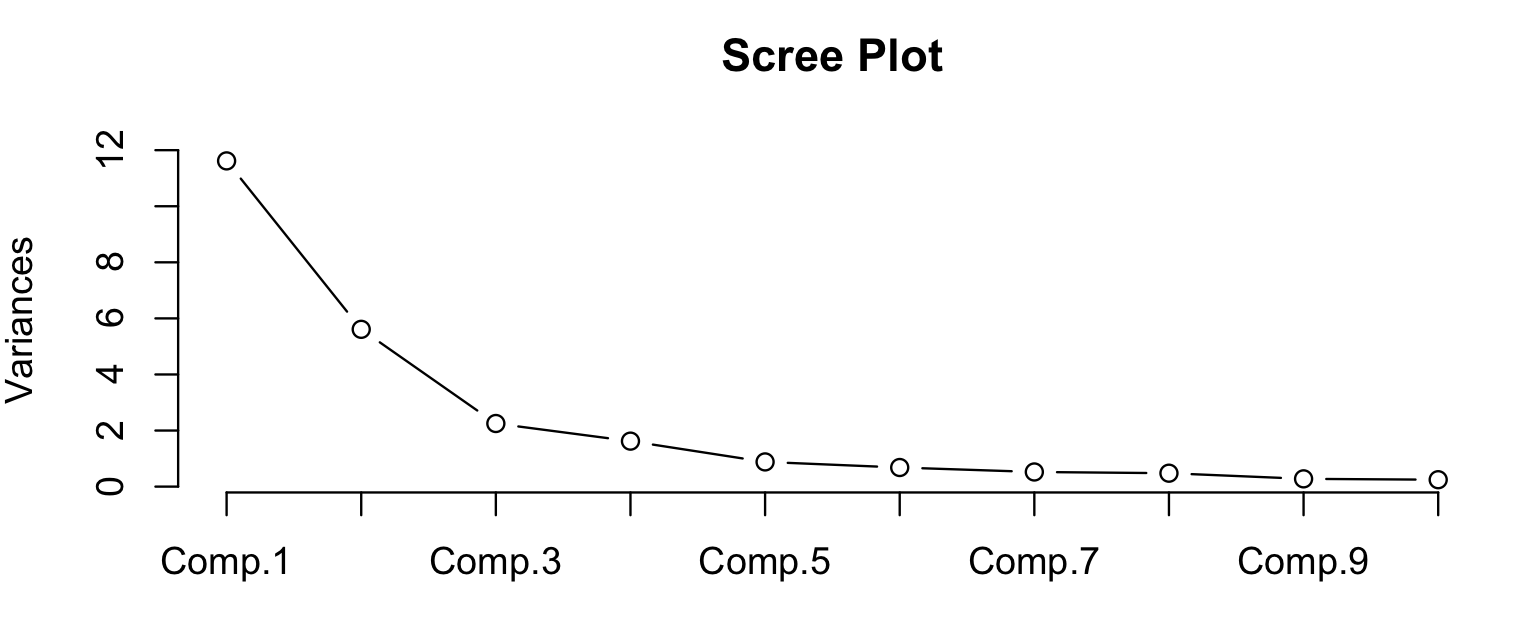


Figure 7: Scree Plot

Table 5: Selected Principle Components

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Component No** | **No.1** | **No.2** | **No.3** | **No.4** | **No.5** | **No.6** | **No.7** | **No.8** | **No.9** |
| Standard deviation | 3.41 | 2.37 | 1.50 | 1.27 | 0.94 | 0.83 | 0.72 | 0.69 | 0.53 |
| Proportion of Variance | 0.46 | 0.22 | 0.09 | 0.06 | 0.04 | 0.03 | 0.02 | 0.02 | 0.01 |
| Cumulative Proportion | 0.46 | 0.69 | 0.78 | 0.84 | 0.88 | 0.91 | 0.93 | 0.95 | 0.96 |

Based on these 9 principle components, a neural network based on MLFF\_BP was rebuilt, containing 20 inputs and one hidden layer with 3 hidden nodes. From Table 6 below, a slight improvement of around 0.5 MSE decrease was gained after dimension reduction. The structure of this improved model shows below in Figure 8.

Table 6: MLFF\_BP Performance

|  |  |  |  |
| --- | --- | --- | --- |
| **Network Stucture** | **training loss** | **validation loss** | **test loss** |
| 36-6-1 | 0.1900 | 0.1820 | 0.1893 |
| 20-3-1 | 0.1857 | 0.1788 | 0.1812 |

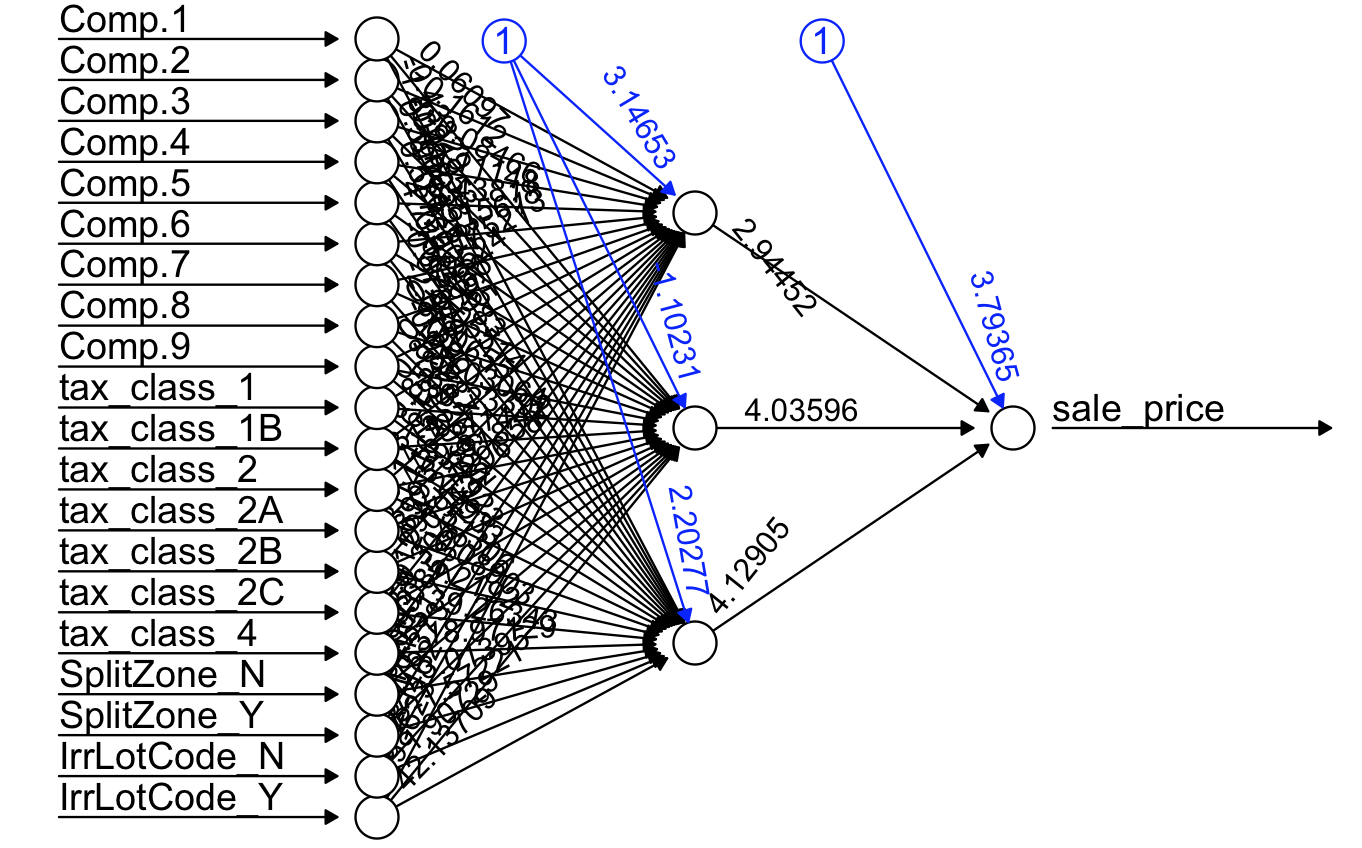


Figure 8: 20-3-1 MLFF-BP Structure

To visualize the predictive performance of the improved model, a scatter plot of predictive and actual sales price is shown as below, from which an acceptable predictive performance can be identified.

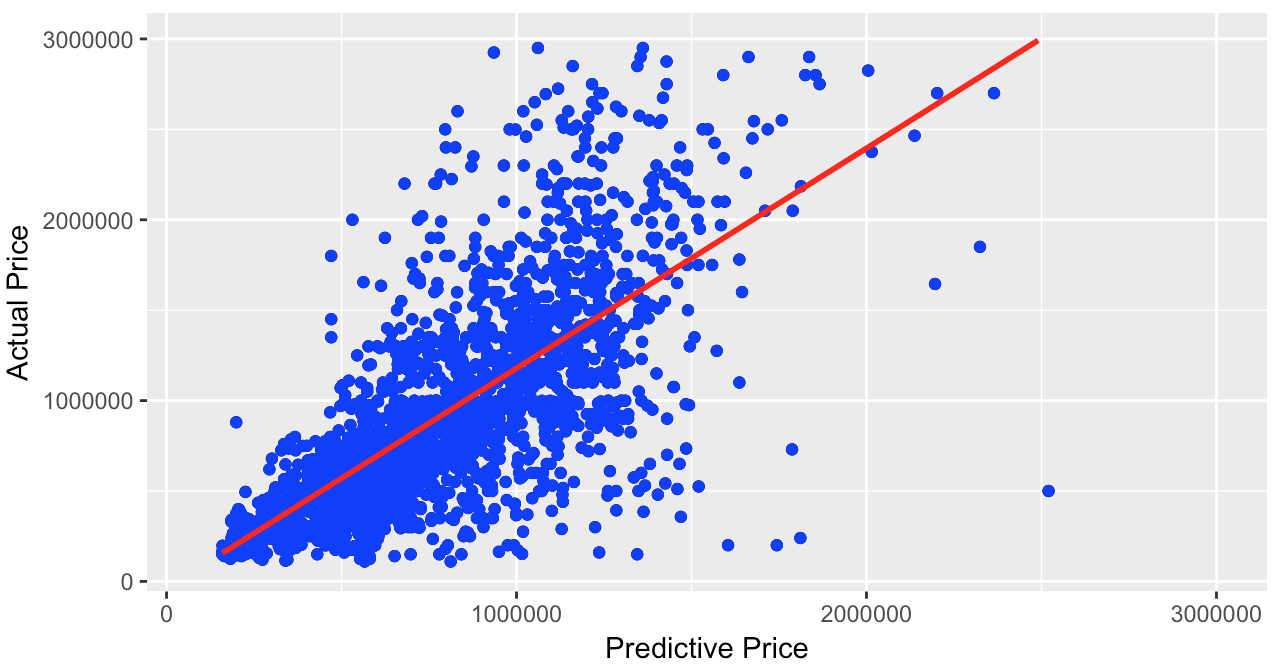


Figure 9: Scatter Plot of Actual and Predictive Value, MLFF-BP, Improved

# Radial Basis Function (RBF) Neural Network Model

## Model Introduction

Radial Basis Function (RBF) is an artificial neural network that uses radial basis functions (most commonly, the Gaussian basis function). Different from MLFF-BP network mentioned in chapter 3, RBF typically have three layers, including an input layer, a hidden layer with a basis function and a linear output layer. This property makes RBF network more locally sensitive and it is good for novelty detection. In addition, compared with MLFF-BP network, RBF can converge faster and good for classification and be more interpretable.

## Model Building

To build a RBF neural network, the following parameters were set as below. Based on empirical experience, when the number of input samples is not large enough, a kernel center is assigned to each input node, as a form of self-growing network. As such, the number of kernels in the hidden layer was set to 52, the same as the number of input nodes. K-means clustering is used to determine the center for each of the radial basis function, by which a weighted sum output can be produced for each given input. Also, the initialization is performed by RBF\_Weights\_Kohonen and RBF-weights as well. As the output is not logistic, the initialization parameters are set normally as c(0, 1, 0, 0.01, 0.01).

Table 7: RBF Properties

|  |  |
| --- | --- |
| **RBF Properties** | **Value** |
| number of hidden nodes | 36 |
| Activation function | Gaussian |
| iterations to learn | 1000 |
| initialization function | Kohenen & RBF weights |
| Initialization function parameters | Normal |

## RBF Model Performance

After 1000 iterations, the training loss of RBF model was 0.5212, while the validation loss was lower due to a relatively smaller sample size. The loss value of the test dataset seems to be the same as the one of training dataset. From such comparison, the RBF model was assumed to have a good predictive ability.

Table 8: MAE of RBF Model

|  |  |  |  |
| --- | --- | --- | --- |
| **iterations** | **training loss** | **validation loss** | **test loss** |
| 1000 | 0.2833 | 0.2716 | 0.2790 |

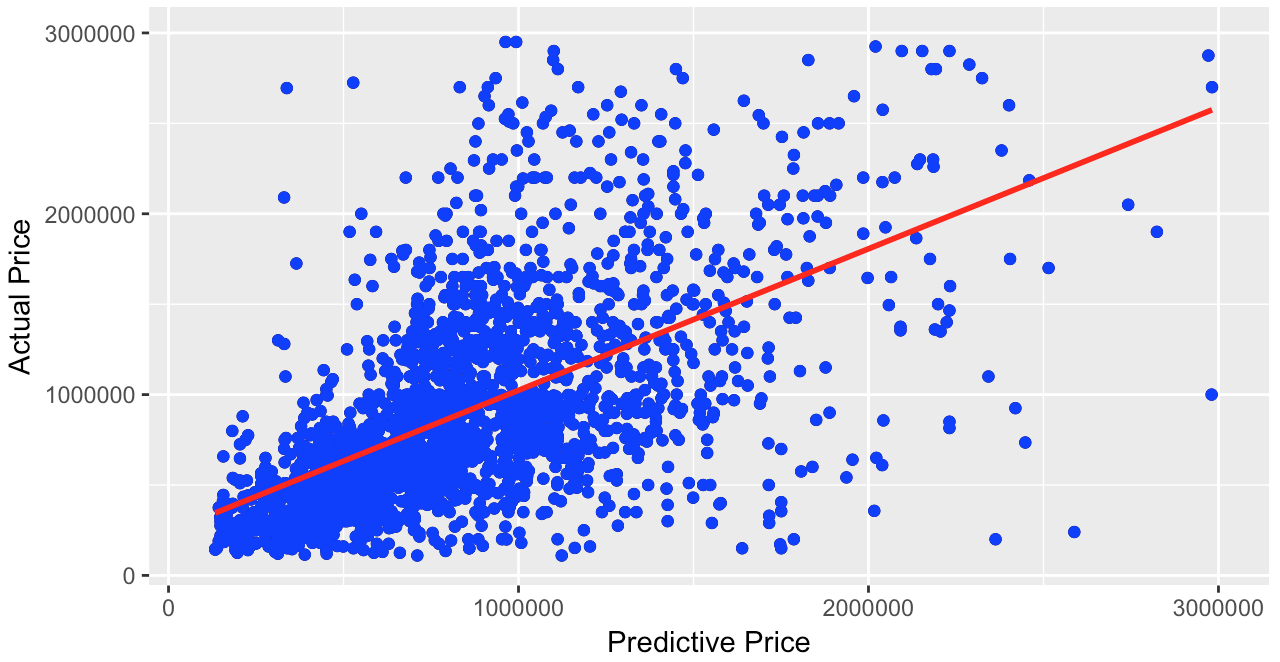


Figure 10: Scatter Plot of Actual and Predictive Value, RBF

# Models Ensemble

Model ensemble is a very powerful technique to increase accuracy of various machine learning models. In addition, the process of ensemble has no need to retrain a model, relying on the predictions on test set for different methods. Therefore, ensemble is a quick way to improve model accuracy.

Since housing price as the target in this case is a continuous variable, Averaging as an ensemble method served well for a wide range of problems including both classification and regression was chosen, the structure shown in Figure 11. It works by taking the mean of each model predictions. As such, an ensemble model frequently performs better than any individual model, as the various errors of different models are averaging out.

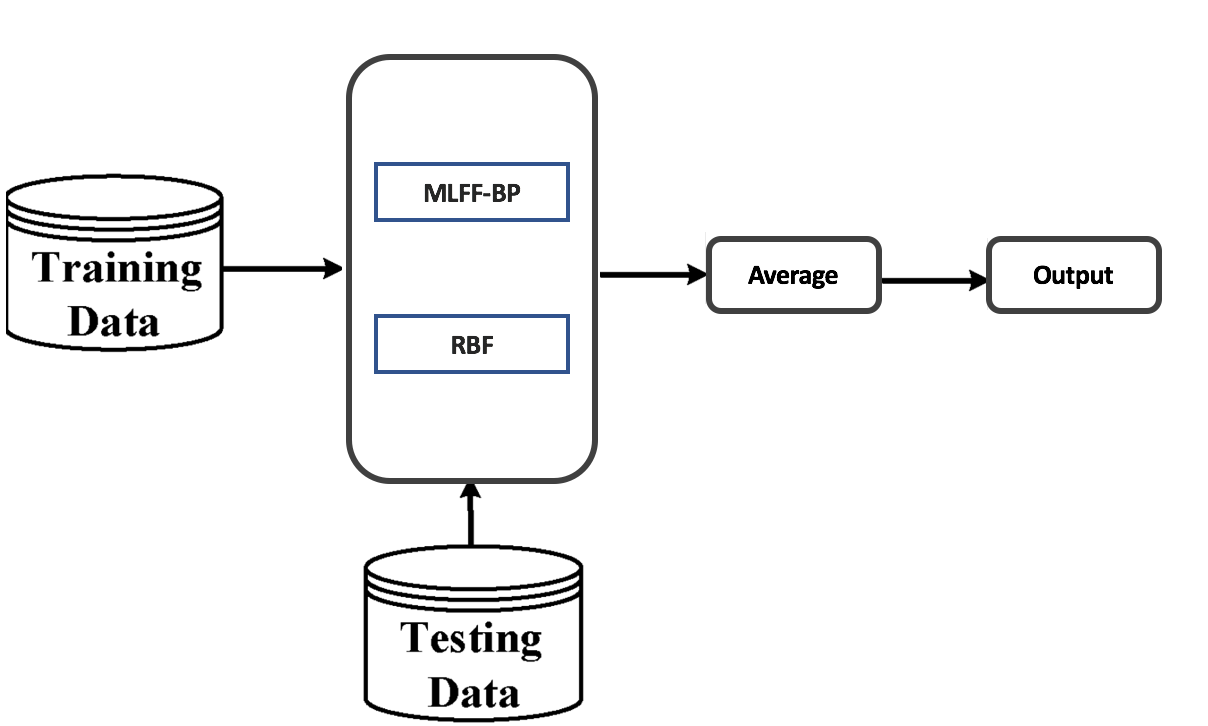


Figure 11: Ensemble Structure

From Table 6 below, a significant improvement can be observed by the loss value of the averaging ensemble model, MAE of the ensemble model is lower than both MLFF-BP and RBF network.

Table 9: MAE of Ensemble Mode by Averaging

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **MLFF-BP loss** | **RBF loss** | **Ensemble loss** |
| Averaging | 0.1812 | 0.2790 | 0.1791 |

In this case, due to the lack of domain knowledge, it is difficult to apply a weighted sum of the prediction of each model, rather than a mere average of all results. It is also noted that most forms of neural networks are some subset of a linear combination. Therefore, from this prospective, there are still more improvement can be achieved to enhance the predictive ability.

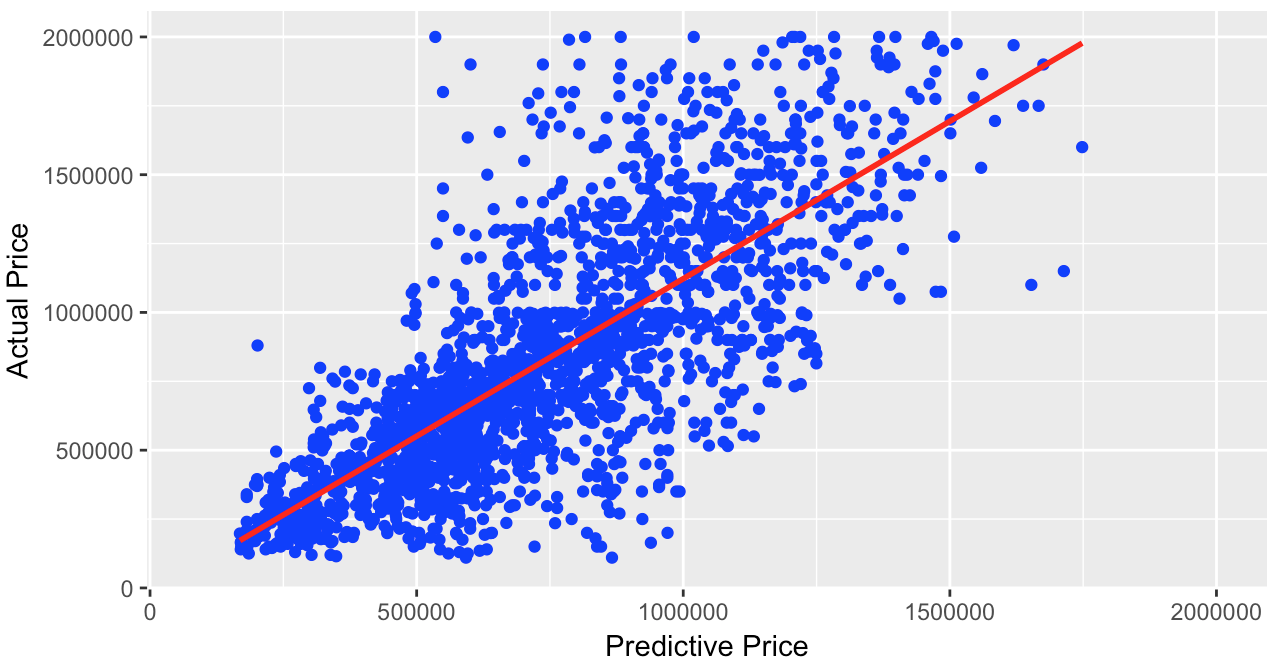


Figure 12: Scatter Plot of Actual and Predictive Value, Ensemble

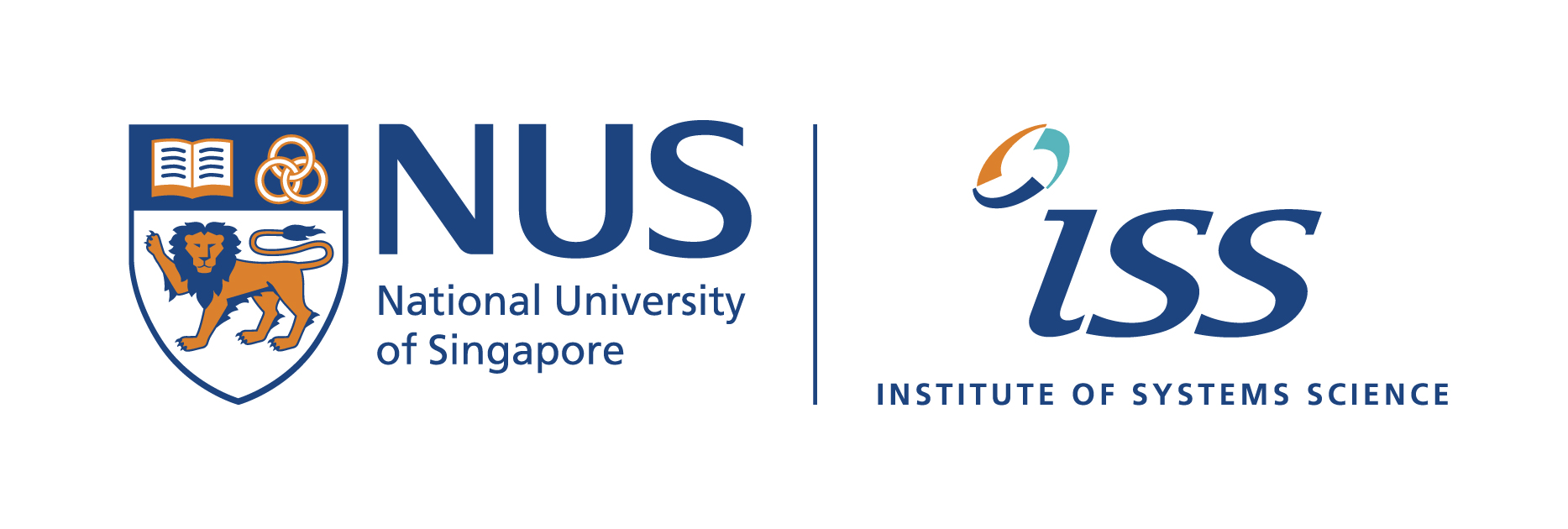
# Conclusion

In this report, a housing price dataset of Ames was used to build a neural network regression model to predict the housing price based on other aspects of residential buildings. To meet the assumptions about inputs of neural networks, the selected variables were transform and standardization into acceptable forms. After that, two neural network predicting models were built according to two algorithms, MLFF-BP and RBF. MSE, as a parameter chosen to assess the model performance, showed similar values in these two models, while the performance of MLFF-BP was relatively better. To decrease MSE value of the model, ensemble was implemented as a solution. Based on averaging ensemble, the error of the test dataset decreased to 0.1791, which was lower than individual model. Furthermore, there are still a large margin of improvement that ensemble methods could achieve, while combining related domain knowledge.

**Appendix A**

[1] The metadata is provided by New York City agencies link: <https://www.kaggle.com/rameshmech/brooklyn-home-sales-prediction-ramesh-dhanapal/data>

[2] Files and codes related to the experiments, we have pushed to the github remote repository, link is <https://github.com/alex44jzy/NUS-CI1-CA1>.

****

**EB5206 Computational Intelligence**:

**Classification Assignment**

**­­­­­­­Facial Expression Recognition Classification**

**Lecturer:** Dr. ZHU Fang Ming

Dr. TIAN Jing

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Name** | **E-mail** |
| A0178431A | Huang Qingyi | [e0267742@u.nus.edu](mailto:e0267742@u.nus.edu) |
| A0178415Y | Jiang Zhiyuan | [e0267726@u.nus.edu](mailto:e0267726@u.nus.edu) |
| A0178222H | Zhu Huiying | e0267533@u.nus.edu |

# Table of Contents

[**Facial Expression Recognition Classification** 15](#_Toc514180109)

[Table of Contents 0](#_Toc514180110)

[**Facial Expression Recognition Classification** 1](#_Toc514180111)

[Abstract 1](#_Toc514180112)

[1. Data Understanding 2](#_Toc514180113)

[2. Data Preparation 3](#_Toc514180114)

[2.1 Check for missing data and outliers 3](#_Toc514180115)

[2.2 Fit format for model 3](#_Toc514180116)

[2.3 Feature-wise standardization 3](#_Toc514180117)

[2.4 Image Augmentation 4](#_Toc514180118)

[2.5 Comments 5](#_Toc514180119)

[3. Build three similar VGG models using Keras 6](#_Toc514180120)

[3.1 Network Structure 6](#_Toc514180121)

[3.2 Parameter Choosing 7](#_Toc514180122)

[3.3 Avoid Overfitting 10](#_Toc514180123)

[3.3.1 Dropout 10](#_Toc514180124)

[3.3.2 Image Augment 10](#_Toc514180125)

[3.4 Experiment Results 11](#_Toc514180126)

[3.5 Comments 12](#_Toc514180127)

[4. Comparison and Ensemble 13](#_Toc514180128)

[4.1 Confusion Matrix 13](#_Toc514180129)

[4.2 Ensemble three models 15](#_Toc514180130)

[4.3 Comments 15](#_Toc514180131)

[5. Conclusion 16](#_Toc514180132)

[6. Future Work 16](#_Toc514180133)

[7. Reference 17](#_Toc514180134)

**­­­­­­­Facial Expression Recognition Classification**

# Abstract

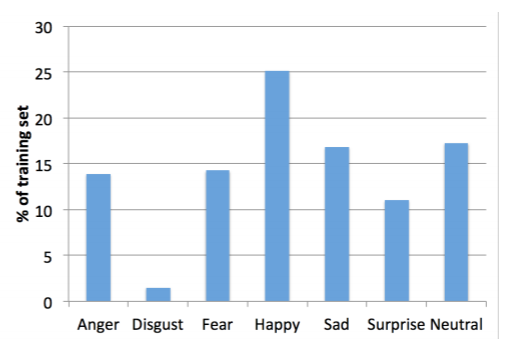
Facial expressions convey non-verbal information between humans in face-to-face interactions. Automatic facial expression recognition, which plays a vital role in human-machine interfaces, has attracted increasing attention from researchers since the early nineties. Automated facial expression recognition has numerous practical applications such as psychological analysis, medical diagnosis, forensics (lie-detection), studying effectiveness of advertisement and so on. Classical machine learning approaches often require a complex feature extraction process and produce poor results. In our experiment, we apply recent advances in deep learning to propose effective deep Convolutional Neural Networks (CNNs) that can accurately interpret semantic information available in faces in an automated manner without hand-designing of features descriptors. We compare different types of CNN models and training tricks in order to learn CNNs with a strong classification power. To achieve better prediction results, we ensemble different models using confusion matrix. The experimental results show that our proposed networks get close to state-of-the-art methods on the well-known FERC-2013 dataset provided on the Kaggle facial expression recognition competition. In comparison to the winning model of this competition, the number of parameters in our proposed networks intensively decreases, that accelerates the overall performance speed and makes the proposed networks well suitable for real-time systems. To run this large dataset, we use GPU on Google Colab, Python 3 and National Supercomputing Centre Singapore (NSCC) using configuration: 24 CPUs, 80G memory and GPU (nVidia Tesla K40).

**Key words:** **VGG, Facial expression recognition, Image augmentation, Adam optimizer, Ensemble.**

# Data Understanding

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression in to one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral) using three tools.

Fer2013.csv contains three columns, "emotion", "pixels" and "usage". The "emotion" column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image. The "pixels" column contains a string surrounded in quotes for each image. The contents of this string a space-separated pixel values in row major order. The "usage" column contains two kinds of strings indicating two usages of the data, "training" and "test". The training set consists of 28,709 examples. The test set used for the leaderboard consists of 3,589 examples. Additionally, the distribution of each kind of expression in the training dataset is shown in Fig. 1.1.



**Fig. 1.1 Distribution of Expressions in Training Dataset**

# Data Preparation

Data preparation has significant effect on results and it determines the performance of models. We did four steps as followed:

## Check for missing data and outliers

The data are intact and each pixel is in range from 0 to 255. As such, there is no missing data and outliers.

## Fit format for model

* For input

Input shape of our model is “channel\*height\*width”. The raw training data has one dimension (1\*2304), so we change each sample to three dimensions ([1\*48\*48]).

* For target

Due to choosing “*categorical\_crossentropy”* as loss function, we should convert targets to categorical format (e.g. targets have 7 classes, the target for each sample should be a 7-dimensional vector that is all-zeros except for a 1 at the index corresponding to the class of the sample)

## Feature-wise standardization

Subtracting the mean centers, the input to 0, and dividing by the standard deviation makes any scaled feature value the number of standard deviations away from the mean.

CNNs learn by continually adding gradient error vectors (multiplied by a learning rate) computed from backpropagation to various weight matrices throughout the network as training examples are passed through, as shown in formula 2.1. If we didn't scale our input training vectors, the ranges of our distributions of feature values would likely be different for each feature, and thus the learning rate would cause corrections in each dimension that would differ (proportionally speaking) from one another. We might be over compensating a correction in one weight dimension while undercompensating in another. To avoid that, we standardise each input feature. The standardized images are shown in Fig. 2.2.

(2.1)

## Image Augmentation

Due to the small amount of training data, we apply data augmentation techniques to increase the amount of training samples in order to avoid overfitting and improve recognition accuracy. Keras provides the “[*ImageDataGenerator*](http://keras.io/preprocessing/image/)” class that defines the configuration for image data preparation and augmentation, Shown as Fig. 2.1.

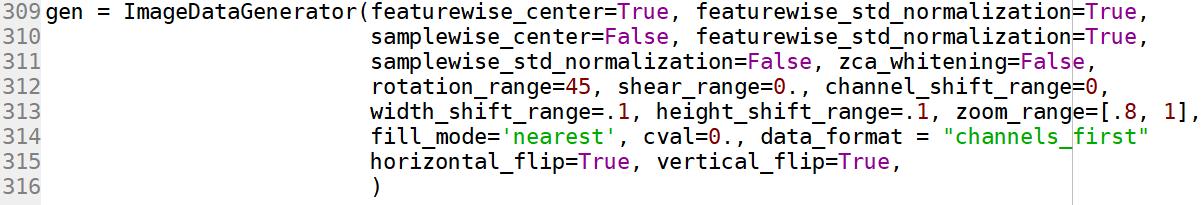
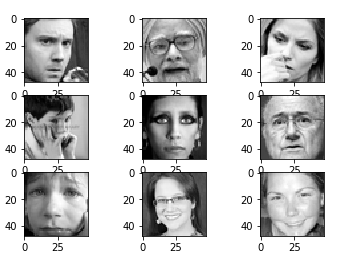
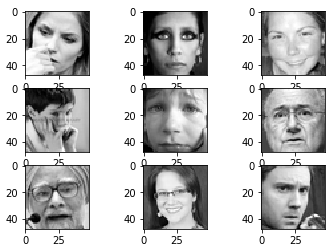


Fig. 2.1 Image Augmentation by ImageDataGenerator in Python

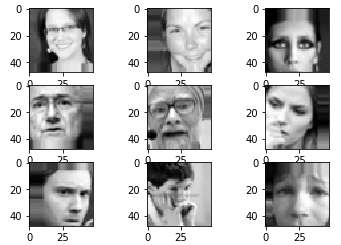
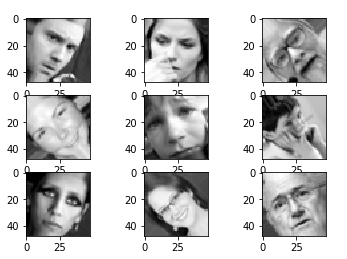
For each image, we perform the following successive transforms:

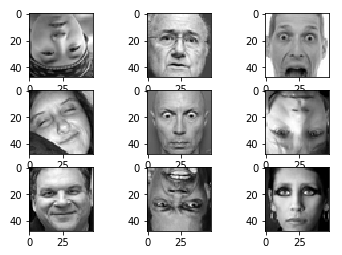
* Rotate the image with a random angle from -45 to 45 (in degrees).
* Shift the width or height of image with a random 20 percent of total width or height.
* Randomly flip images horizontally or vertically.

Then we compare the images after processing with originals shown in Fig. 2.2 with random order.



1. **Original image. (b) Standardized images**



 **(c) Rotation range = 45° (d) Shift = 0.2**

**(e) Randomly flip**

**Fig. 2.2 Images comparison before and after processing**

From Fig. 2.2, we find that there is no obviously observed change between original and standardised images. And we exploit augmentation techniques to create more data for training. Moreover, we will compare the accuracy rate between augmentation and not augmentation in sector 3.3.2.

## Comments

In this sector, we preceed suitable data pre-processing work, including handling missing data and outliers, fitting format for model, normalization and image augmentation, which can promote models performance dramatically. After this, we can build models in sector 3.

# Build three simplify VGG models using Keras

With the fast growth of deep learning, the state-of-the-art in many computer vision tasks has been considerably improved. In image classification, there are some well-known deep CNNs can be mentioned as follows. The first network we want to mention is AlexNet, the winner of ImageNet ILSVRC challenge in 2012. This network has a very similar architecture to LeNet, but is deeper and bigger, and its convolutional layers stack on top of each other. Previously it is common to have only a single convolutional layer followed by a pool layer. The next CNNs are called VGGNet, the runner-up in ILSVRC 2014. One important property of VGGNet is that there are many convolutional layers with small filter size 3 × 3 that stack on top of each other instead of using a single convolutional layer with larger filter size as in previous CNN generations. The winner of ILSVRC 2015 is ResNet[2] which can be characterized by skip connections and heavy use of batch normalization. Recent improved variants of ResNet called Wide ResNet or ResNeXt also demonstrate their impressive power in image classification tasks.

## Network Structure

VGGNet[3] has simple designing idea of architecture and has better results in the problem of facial expression recognition, and we refer to a thesis “Facial Expression Recognition Using Deep Convolutional Neural Networks”[1], following their models, we build three simplify VGG(S-VGG) models shown as Table 3.1. Then we compare their performance in Section 3.3.

**Table 3.1 Three S-VGG model architectures**

|  |  |  |
| --- | --- | --- |
| ConvNet Configuration | | |
| S-S-VGG10 | S-S-VGG12 | S-S-VGG14 |
| 10 layers | 12 layers | 14 layers |
| Input (1 \* 48 \* 48) | | |
| Conv3 - 32 | Conv3-32  Conv3-32 | Conv3-32  Conv3-32 |
| maxpool | | |
| Conv3-64 | Conv3-64  Conv3-64 | Conv3-64  Conv3-64 |
| maxpool | | |
| Conv3-128  Conv3-128 | Conv3-128  Conv3-128 | Conv3-128  Conv3-128  Conv3-128 |
| maxpool | | |
| Conv3-256  Conv3-256  Conv3-256 | Conv3-256  Conv3-256  Conv3-256 | Conv3-256  Conv3-256  Conv3-256  Conv3-256 |
| FC-256 | | |
| FC-256 | | |
| FC-7 | | |

All of these architectures follow the general designing principles of VGG. They differ from each other only in depth: from 10 layers in S-VGG10 (7 convolutional and 3 fully connected layers) up to 14 layers in S-VGG14 (11 convolutional and 3 fully connected layers).

For example, S-VGG12 is described in more details as follows. Overall, S-VGG12 consists of four blocks. The first block has two 3 × 3 convolutional layers, the number of filters is 32, the stride is 1. The second block has two 3 × 3 convolutional layers, the number of filters is 64, the stride is 1. The third block has two 3×3 convolutional layers, the number of filters is 64, the stride is 1. The fourth block has three 3×3 convolutional layers, the number of filters is 64, the stride is 1. After each block except the last one, there is a max pooling layer with filter size 2 × 2 and the stride is 2. After the four blocks, there are two fully connected layers with 256 neurons per each layer. Finally, the output layer is a fully connected layer with 7 neurons associated with 7 emotion classes. The softmax activation function is applied to output layer and cross-entropy loss function is used in training process.

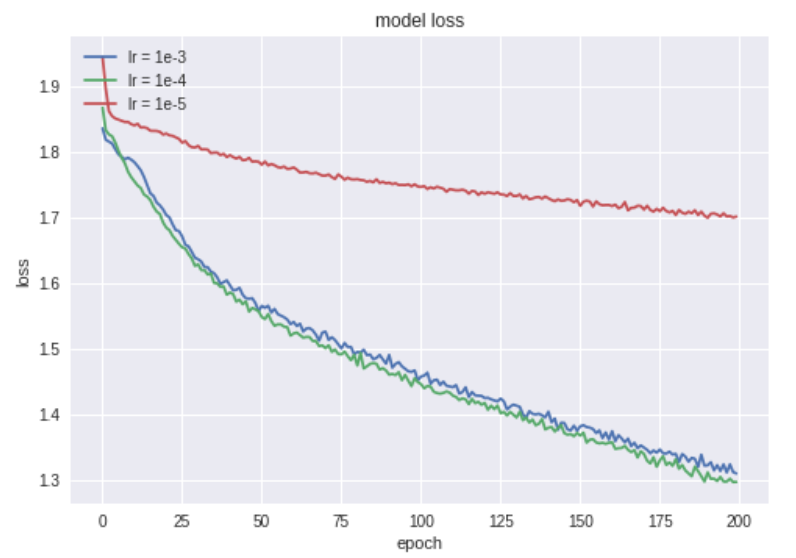
## Parameter Choosing

In section 3.2 we use S-VGG10 model to compare different parameters performance, and then explore the optimum parameters in S-VGG12 and S-VGG 14.

**3.2.1 Optimizer**

Compared three popular optimizers Adam, Adadelta and RMSprop, we find Adam works well in practice and compares favourably to other adaptive learning-method algorithms as it converges very fast and the learning speed of the Model is quiet Fast and efficient. Moreover, it rectifies every problem that is faced in other optimization techniques such as vanishing Learning rate, slow convergence or High variance in the parameter updates which leads to fluctuating Loss function. As such, we choose Adam as optimizer.

Then we test different learning rate effect, shown in Fig. 3.1. After 200 epochs, learning rate equal to 0.0001 has the lowest loss, so we choose 0.0001 as learning rate. By setting that learning rate, the model can cost less study time and acquire the best convergence result.



**Fig. 3.1 Model loss based on various learning rate**

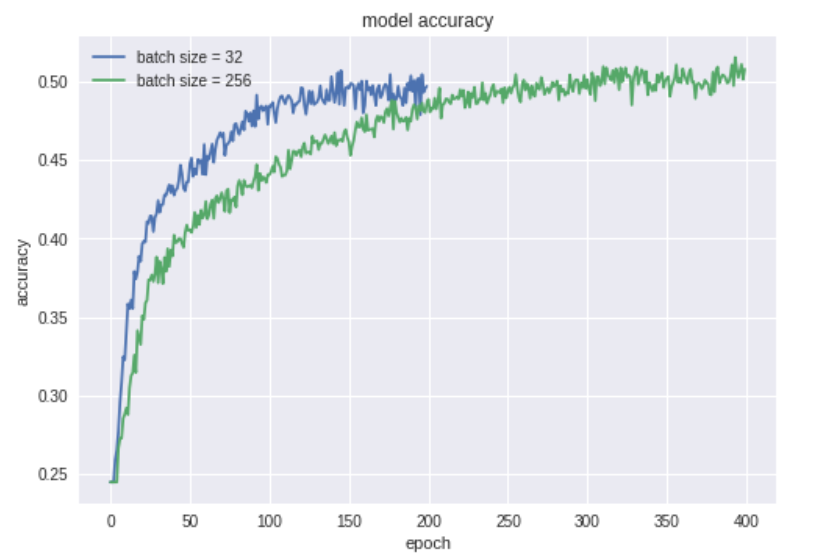
**3.2.2 Batch Size**

To train all the data in each epoch, the product of batch size and steps\_per\_epoch must be greater than length of training dataset, so we set **steps\_per\_epoch=len(x\_train)//batch\_size** and get different time consuming per epoch among different training batch size, shown as table 3.2.

**Table 3.2 Time consuming among different batch size**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Batch Size | 10 | 32 | 256 | 480 |
| Seconds/epoch | 74 | 26 | 12 | 11 |

Delete 10 batch size for quick training. And with batch size increasing, the number of epochs is also required higher to convergence. Based on the two points, we finally compare model validation accuracy among 32 and 256 training batch size and the results are shown as Fig. 3.2 and Fig. 3.3.



**Fig. 3.2 Model loss based on various training batch sizes**



**Fig. 3.3 Model loss based on various training batch sizes**

When training in 32 batch size, the model converges after 200 epochs, and when training in 256 batch size the model converges after 250 epochs. So we can get conclusion that the larger batch size, the more epochs needed in the Fig. 3.3. But when training in 256 batch size, model has the lowest validation loss and higher accuracy, so we choose 256 as training batch size.

## Avoid Overfitting

### Dropout

To avoid overfitting[4], we drop different 20% and 50% in fully connected layers, and compare the accuracy rate with none dropout, shown in table 3.3.

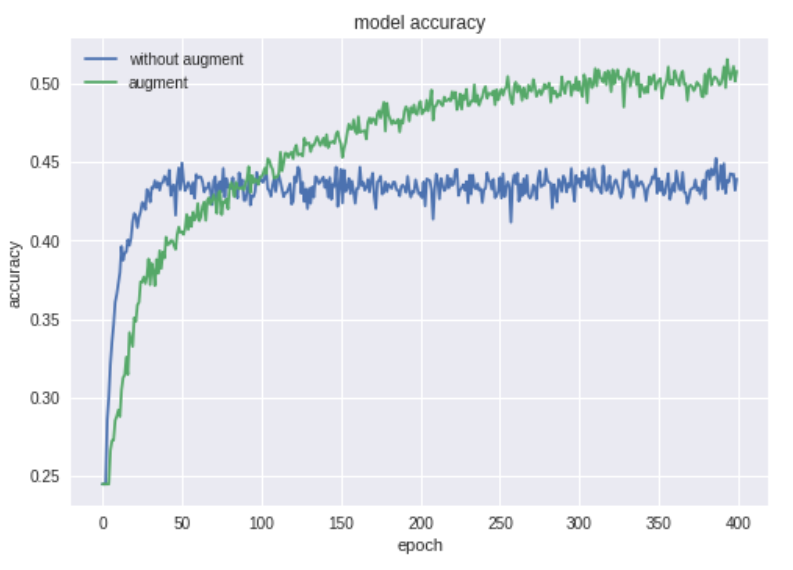
**Table 3.3 S-VGG10 test accuracy by different dropout**

|  |  |  |  |
| --- | --- | --- | --- |
| Dropout | None | 20% | 50% |
| Accuracy(percent) | 49.97 | 50.73 | 52.36 |

From table 3.3, we find 50% dropout has highest accuracy, so we adopt 50% dropout in later models.

### Image Augment

As mentioned in sector 2.4, we exploit Image Augment[5] to create more data for training, then we draw accuracy- epoch curves in training dataset and test dataset, shown in Fig. 3.4 and Fig. 3.5. From this Fig., we get Image Augment can promote the model performance.



**Fig. 3.4 Model accuracy with augmentation vs without augmentation**



**Fig. 3.5 Model loss augmentation vs without augmentation**

Another point we find is that training without augment only need 5 seconds per epoch vs 12 seconds per epoch with augment. Although the time is less, but result is not so good. Therefore, we adopt image augment to later models.

## Experiment Results

Based on optimised parameters, we build S-VGG10, S-VGG12 and S-VGG14 and get their loss and accuracy curve, shown in table 3.4. S-VGG10 to S-VGG14 the model becomes more and more complex thanks to increasing depth and can fit better the dataset.

**Table 3.4 Model Accuracy**

|  |  |  |  |
| --- | --- | --- | --- |
| Model | S-VGG10 | S-VGG12 | S-VGG14 |
| Accuracy | 52.36 | 53.59 | 56.36 |

In table 3.5, we compare the model complexity of our model with the champion team of the competition. Although the result of the campion team is 69.4% on the test dataset, our models have much smaller number of parameters, so our results can be acceptable.

**Table 3.5 Number of parameters**

|  |  |
| --- | --- |
| Architecture | Number of parameters |
| Champion team’s model | 7.17M |
| S-VGG10 | 4.14M |
| S-VGG12 | 4.19M |
| S-VGG14 | 4.92M |

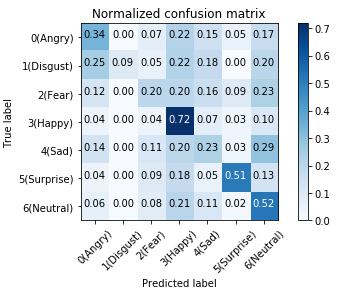
## Comments

In this sector, we mainly optimize 2 parameters batch size and learning rate, which are based on models convergence speed and validation accuracy. Furthermore, utilize adding dropout layers and image augmentation aimed to effectively avoid model overfitting. Compared with the champion model in Kaggle competition, our 3 models own less parameters volume and acquire the equivalent performance on the aspect of accuracy and loss. In addition, with enhancing the depth of model layers, the performance of S-VGG14 seems better than the reset 2 S-VGG models with shallow layers.

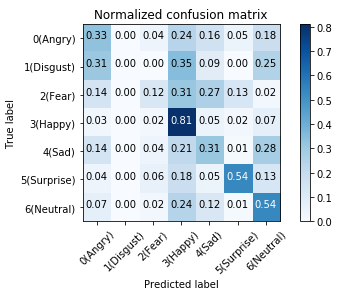
# Comparison and Ensemble

## Confusion Matrix

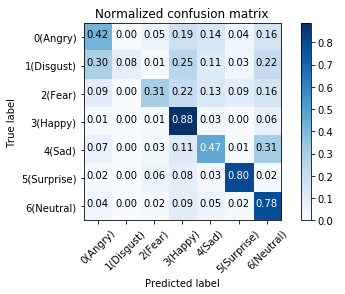
Fig. 4.1 presents the confusion matrix of recognition result achieved by S-VGG10, S-VGG12 and S-VGG14 architecture. As convolutional layers are deeper, each kind of expression accuracy is increasing. High accuracies are obtained with happy (88%), surprised (80%), and neutral (78%) classes. In fact, they are the most distinguishable emotions for human. The angry, fearful, sad classes are more often confused together, since they share many similar expressions. Finally, the disgusted class get a relatively good accuracy 9%, despite the low amounts of disgusted samples in the training set.



1. **S-VGG10**



**(b) S-VGG12**



**(c) S-VGG14**

**Fig. 4.1 Normalized Confusion Matrix of S-VGG10, S-VGG12, S-VGG14**

## Ensemble three models

To get higher accuracy, we follow the rule shown in Fig. 4.2 to make the three VGG models ensemble.



**Fig. 4.2 Model ensemble block diagram**

The final ensemble accuracy is shown in table 4.1.

**Table 4.1. Ensemble accuracy of 7 expression and the ensemble average accuracy**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Type | Angry | Disgust | Fear | Happy | Sad | Surprise | Neutral | **Average** |
| Accuracy(%) | 45.23 | 11.73 | 39.88 | 89.56 | 50.77 | 79.77 | 75.36 | **59.34** |

## Comments

In this section, we build S-VGG10, S-VGG12 and S-VGG14. The accuracy is higher as convolutional layer are deeper. Moreover, we adopt ensemble to achieve better result. We find not only average accuracy is higher, each type of expression accuracy is also higher, so ensemble model has an significant influence on the performance of the whole system.

# Conclusion

In this project, inspired by the designing principles of VGG, we adopt effective architectures of CNNs to tackle the problem of facial expression recognition. The proposed networks are composed of a stack of convolutional blocks. Each block has a few 3 × 3 convolutional layers followed a max pooling layers. Despite having much smaller number of parameters, the model becomes more and more complex thanks to increasing depth and can fit better the dataset. The results prove the power of small filter and very deep network in classification tasks. We also show data augmentation is be an important trick in training deep neural network. Additionally, ensemble models can promote the final accuracy.

Files and codes related to the experiments, we have pushed to the github remote repository url is <https://github.com/alex44jzy/NUS-CI1-CA1>. And the facial expression dataset is sourced from Kaggle competition (Challenges in Representation Learning: Facial Expression Recognition Challenge), link is <https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge>.

# Future Work

Here, we propose two approaches to further improve the performance of our Simplify VGG neural networks as our future works.

As the percentage of disgust expression images among the whole dataset is only 2%, very less, and they are quite similar to angry expression images, as a result the accuracy of disgust is relatively low (11.73 %). Such a low accuracy has influenced the overall testing accuracy. So we plan to investigate how to improve the testing accuracy by concentrating on the disgusting expression data.

Firstly, since the training images with label of disgusting expression accounts for a quite small percentages, we hypothesize that if we remove the small group of images from the training and testing dataset, we would acquire a higher testing accuracy. We believe that the disgusting group images are noises for the dataset, because they are very similar to and always being wrongly classified as “happy”, “angry” and “neutral” (Fig. 4.1).

Secondly, we suggest to manually increase the percentage of the small disgusting expression images group by data augmentation technology. By the reason that the training samples are too less, the neural network might not converge for the disgusting expression images. So if we increase the number of samples from this label group, it is expected to give out a better testing accuracy.

# Reference

[1] Sang, D.V. and Van Dat, N., 2017, October. Facial expression recognition using deep convolutional neural networks. In Knowledge and Systems Engineering (KSE), 2017 9th International Conference on (pp. 130-135). IEEE.

[2] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[3] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[4] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), pp.1929-1958.

[5] Wong, S.C., Gatt, A., Stamatescu, V. and McDonnell, M.D., 2016, November. Understanding data augmentation for classification: when to warp?. In Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on (pp. 1-6). IEEE.